THE CARE AND FEEDING OF LR(k) GRAMMARS[†]

Alfred V. Aho
Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey

and

Jeffrey D. Ullman
Princeton University
Princeton, New Jersey

Abstract

We consider methods of modifying LR(k) parsers [1] while preserving the ability of that parsing method to detect errors at the earliest possible point on the input. Two transformations are developed, and the methods of Korenjak [2] and DeRemer [3] are expressed in terms of these transformations. The relation between these two methods is exposed. Proofs are for the most part omitted, but can be found in [4].

I. Introduction

The LR(k) grammars [1] are the grammars which can be parsed bottom-up, deterministically, by a deterministic pushdown automaton which acts in a "natural" way, i.e., by finding substrings in right sentential forms which match right sides of productions and replacing these substrings by left sides. We therefore begin with a definition of a context free grammar and LR(k)-ness.

A context free grammar (CFG) is a four-tuple $G = (N,\Sigma,P,S)$, where N and Σ are finite disjoint sets of nonterminals and terminals, respectively; S, in N, is the start symbol, and P is a finite set of productions of the form $A \rightarrow \alpha$, where A is in N and α in $(N \cup \Sigma)^*$. We assume productions are numbered $1,2,\ldots,p$ in some order.

Conventionally, A, B and C denote nonterminals, a, b and c denote terminals and X, Y and Z are in $N \cup \Sigma$. We use $u,v,\ldots,z$ for strings in Σ^* and $\alpha,\beta,\ldots$ for strings in $(N \cup \Sigma)^*$. We use e for the empty string.

If $A \rightarrow \alpha$ is in P, then for all β and γ we write $\beta A\gamma \Rightarrow \beta\alpha\gamma$. If γ is in Σ^*, then the replacement is said to be rightmost, and we write $\beta A\gamma \Rightarrow \beta\alpha\gamma$. The explicitly shown α is said to be a handle of $\beta\alpha\gamma$ in this event. $\overset{*}{\Rightarrow}$ and $\underset{rm}{\overset{*}{\Rightarrow}}$ denote the reflexive and transitive closure of $\Rightarrow$ and $\underset{rm}{\Rightarrow}$, respectively. Arrows may be subscripted by the name of the grammar, to resolve ambiguities. The language defined by G is $L(G) = \{w | S \overset{*}{\Rightarrow} w\}$. It is known that if $S \overset{*}{\Rightarrow} w$, then $S \underset{rm}{\overset{*}{\Rightarrow}} w$. That is, every sentence in the language has a rightmost derivation. A right sentential form of G is a string α such that $S \underset{rm}{\overset{*}{\Rightarrow}} \alpha$. If $S \underset{rm}{\overset{*}{\Rightarrow}} \alpha Aw \underset{rm}{\Rightarrow} \alpha\beta w$, then γ, a prefix of $\alpha\beta$, is called a viable prefix of G.

We define $\text{FIRST}_k^G(\alpha)$ as $\{w | \alpha \overset{*}{\Rightarrow} wx$ and either $|w|$[††] $= k$ or $|w| < k$ and $x = e\}$. If α is in Σ^*, then $\text{FIRST}_k^G(\alpha)$ has one member, w, which is either α, if $|\alpha|$ is less than k, or the first k symbols of α. In this case we write $\text{FIRST}_k^G(\alpha) = w$ instead of $\{w\}$. Note that $\text{FIRST}_k^G(\alpha)$ is independent of G in this case. We delete G and k from FIRST when no ambiguity arises.

CFG $G = (N,\Sigma,P,S)$ is said to be LR(k) if given the two rightmost derivations

(1) $S \underset{rm}{\overset{*}{\Rightarrow}} \alpha Aw \underset{rm}{\Rightarrow} \alpha\beta w$, and

(2) $S \underset{rm}{\overset{*}{\Rightarrow}} \gamma Bx \underset{rm}{\Rightarrow} \alpha\beta y$,

and $\text{FIRST}_k(w) = \text{FIRST}_k(y)$, then we may conclude that $\gamma Bx = \alpha Ay$.

Informally, G is LR(k) if the unique handle of a right sentential form can be determined by examining that string up to k symbols beyond the right end of the handle.

A consequence of the LR(k) definition is that for each LR(k) grammar $G = (N,\Sigma,P,S)$, a finite set of tables[§] can be constructed. A table is a pair of functions (f,g), where:

[†] This work was partially supported by NSF grant GJ-465.

[††] $|\alpha|$ stands for the length of α.
[§] A set of tables is called a "table" in [1].

(1) f, the parsing action function, maps lookahead strings, i.e., strings in Σ^* of length at most k, to the actions shift, error, accept, and reduce i, where i is the number of a production:

(2) g, the goto function, maps $N \cup \Sigma$ to the set of tables and the word error.

The tables are used to parse input strings bottom up as follows. A pushdown list holds a string of alternating grammar symbols and table names, say $T_0X_1T_1 \ldots X_mT_m$, where the T's are tables and the X's are symbols in $N \cup \Sigma$. If w is the unexpended suffix of the original input string, then $X_1 \ldots X_m w$ will always be a right sentential form if error has not been reported.

Let us suppose we observe the parser at a time when either $m = 0$, i.e., at the beginning, or after it has just performed a reduction, i.e., when X_m is a nonterminal. Let $T_m = (f,g)$. We find $u = \mathrm{FIRST}_k(w)$ and apply function f to u. If that action is shift, the first symbol of u, say a, is shifted onto the pushdown list, and then the table g(a) is shifted on top of a. The pushdown list thus becomes $T_0X_1T_1 \ldots X_mT_maT_{m+1}$, where $T_{m+1} = g(a)$. In the same manner, we shift additional symbols from input to pushdown list, as long as the action of the current table on top of the pushdown list applied to the first k remaining input symbols is shift. The actions error and accept have the obvious meaning.

Eventually, if we are dealing with a right sentential form, we will enter a configuration with $T_0X_1T_1 \ldots X_rT_r$ on the pushdown list and lookahead string x such that the action of T_r on x is reduce i. At that time, the right side of production i will be a suffix of the grammar symbols on the pushdown list, that is, $X_nX_{n+1} \ldots X_r$ for some n. These symbols and the intervening tables are removed from the pushdown list, leaving T_{n-1} on top. Let $T_{n-1} = (f', g')$, let production i have A on the left, and let $g'(A) = T$. Then the symbols A and T are placed on the pushdown list, leaving $T_0X_1T_1 \ldots X_{n-1}T_{n-1}AT$.

We are now "back where we started," with a nonterminal as the top grammar symbol. The process repeats, until either an error or accept action is called for.

The LR(k) tables are constructed from a CFG $G = (N,\Sigma,P,S)$ by first generating what we call sets of items†. An LR(k) item for G is a pair $[A \to \alpha\cdot\beta, u]$, where $A \to \alpha\beta$ is a production and u is in Σ^*, with $|u| \leq k$. Item $[A \to \alpha\cdot\beta, u]$ is said to be valid for γ, a viable prefix of G, if there exists a derivation $S \overset{*}{\underset{rm}{\Rightarrow}} \delta Aw \underset{rm}{\Rightarrow} \delta\alpha\beta w$, where $\gamma = \delta\alpha$, and $u = \mathrm{FIRST}_k(w)$.

The key to constructing LR(k) tables for an LR(k) grammar G is to first construct the sets of valid items for the viable prefixes of G. The number of such sets of items is clearly finite.

Let $G = (N,\Sigma,P,S)$ be a CFG. We define $\mathrm{EFF}_k^G(\alpha)$, or $\mathrm{EFF}(\alpha)$, where G and k are understood, as $\{w \mid \alpha \overset{*}{\underset{rm}{\Rightarrow}} wx$, where $|w| = k$ or $|w| < k$ and $x = e$, and the last step in the derivation $\alpha \overset{*}{\underset{rm}{\Rightarrow}} wx$, if it exists, does not replace the leftmost symbol of the string by e$\}$.†

Let $\mathcal{a}$ be a set of items. The closure of $\mathcal{a}$ is defined to be the least set $\mathcal{a}'$, satisfying:

(1) $\mathcal{a} \subseteq \mathcal{a}'$;

(2) If $[A \to \alpha\cdot B\beta, u]$ is in $\mathcal{a}'$, then $[B \to .\gamma, v]$ is in $\mathcal{a}'$, for all $B \to \gamma$ in P and v in $\mathrm{FIRST}(\beta u)$.

Let $\mathcal{a}$ be a set of items. Then the function $\mathrm{GOTO}(\mathcal{a},X)$, is defined as follows. Let $\mathcal{a}'$ be the set of items $[B \to \alpha X\cdot\beta, u]$ such that $[B \to \alpha\cdot X\beta, u]$ is in $\mathcal{a}$. That is, find all X's to the right of the dot and shift it over. Then $\mathrm{GOTO}(\mathcal{a},X)$ is the closure of $\mathcal{a}'$.

Lemma 1: (From [1]) If $\mathcal{a}$ is the set of items valid for α, then $\mathrm{GOTO}(\mathcal{a},X)$ is the set of items valid for αX.

The set $\mathcal{S}$ of the sets of valid items for an LR(k) grammar $G = (N,\Sigma,P,S)$ can be computed as follows.

(1) Let $\mathcal{a}_0$ be constructed by taking the closure of the single item $[S' \to .S, e]$, where S' is a new symbol.

(2) Begin with $\mathcal{S} = \{\mathcal{a}_0\}$. Repeatedly do step (3).

(3) If $\mathcal{a}$ is in $\mathcal{S}$, add $\mathrm{GOTO}(\mathcal{a},X)$ to $\mathcal{S}$ for all X in $N \cup \Sigma$, if $\mathrm{GOTO}(\mathcal{a},X)$ is nonempty.

Example 1: Consider the LR(1) grammer with productions:

(1) $S \to AS$

(2) $S \to b$

† Partial states in [1].

† EFF stands for e-free first.

(3) $A \rightarrow Aa$

(4) $A \rightarrow b$

The six sets of items for G are listed below. Brackets are removed and the notation $A \rightarrow .Aa$, a/b is short for the two items $A \rightarrow .Aa$, a and $A \rightarrow .Aa$, b, etc.

$\mathcal{a}_0$	$S' \rightarrow .S,e$
	$S \rightarrow .AS,e$
	$S \rightarrow .b, e$
	$A \rightarrow .Aa,a/b$
	$A \rightarrow .b,a/b$
$\mathcal{a}_1$	$S' \rightarrow S.,e$
$\mathcal{a}_2$	$S \rightarrow A.S,e$
	$A \rightarrow A.a,a/b$
	$S \rightarrow .AS,e$
	$S \rightarrow .b,e$
	$A \rightarrow .Aa,a/b$
	$A \rightarrow .b,a/b$
$\mathcal{a}_3$	$S \rightarrow b.,e$
	$A \rightarrow b.,a/b$
$\mathcal{a}_4$	$S \rightarrow AS.,e$
$\mathcal{a}_5$	$A \rightarrow Aa.,a/b$

To see that these are all the sets generated by the above algorithm, we list the GOTO function.

$\mathcal{a}$ \ X	A	S	a	b
$\mathcal{a}_0$	$\mathcal{a}_2$	$\mathcal{a}_1$	-	$\mathcal{a}_3$
$\mathcal{a}_1$	-	-	-	-
$\mathcal{a}_2$	$\mathcal{a}_2$	$\mathcal{a}_4$	$\mathcal{a}_5$	$\mathcal{a}_3$
$\mathcal{a}_3$	-	-	-	-
$\mathcal{a}_4$	-	-	-	-
$\mathcal{a}_5$	-	-	-	-

Fig. 1 GOTO($\mathcal{a}$,X)

Note that the set of valid items for a string which is not a viable prefix is empty. □

The following algorithm constructs a table from a set of items. Let $\mathcal{a}$ be a set of items such that if $[A \rightarrow \alpha.,u]$ and $[B \rightarrow \beta\cdot\gamma,v]$ are in $\mathcal{a}$, then u is not in EFF(γv). Note that γ may be e. (It is known [1] that the sets of items for an LR(k) grammar satisfy this property.) The table (f,g) constructed from $\mathcal{a}$ is defined as follows.

(1) If $[A \rightarrow \alpha.,u]$ is in $\mathcal{a}$, and $A \rightarrow \alpha$ is not $S' \rightarrow S$, then f(u) = reduce i.

(2) If $[A \rightarrow \alpha\cdot\beta,u]$ is in $\mathcal{a}$, $\beta \neq e$, then f(v) = shift for all v in EFF(βu).

(3) If $[S' \rightarrow S.,e]$ is in $\mathcal{a}$, then f(e) = accept.

(4) f(y) = error otherwise.

(5) g(X) is the name of the table constructed from GOTO($\mathcal{a}$,X) wherever GOTO($\mathcal{a}$,X) is not empty.

(6) g(X) = error if GOTO($\mathcal{a}$,X) is empty.

We will refer to the set of tables constructed by this process for an LR(k) grammar as the "Knuth tables."

Example 2: We display the Knuth tables for Example 1 in Fig. 2. Tables throughout this paper are shown as rows, with columns for the arguments of f and g. The following code is used:

X = error
A = accept
S = shift
i = reduce i

	f			g			
	a	b	e	A	S	a	b
T_0	S	S	X	T_2	T_1	X	T_3
T_1	X	X	A	X	X	X	X
T_2	S	S	X	T_2	T_4	T_5	T_3
T_3	4	4	2	X	X	X	X
T_4	X	X	1	X	X	X	X
T_5	3	3	X	X	X	X	X

Fig. 2 Knuth Tables

II. The General Notion of a Set of LR(k) Tables

The amount of computation required to produce a set of LR(k) tables and the number of tables produced using the methods above can be quite large for practical grammars. Recently, several methods [2,3] have been advanced for generation and alteration of a set of LR(k) tables. In order to study these methods, we define an abstract notion of a set of tables, define certain operations on them and show how the transformations of [2,3] can be expressed in terms of these operations.

One preliminary observation is necessary. It is possible that certain entries in an LR(k) table will never be "exercised," that is, they could be replaced with no effect on the parser's operation. We will therefore allow a new action φ, or "don't care" in the range of both f and g.

With this in mind, we define a set of LR(k) tables for a grammar $G = (N,\Sigma,P,S)$ to be a set of pairs of functions $T = (f,g)$, such that

(1) f maps strings u in Σ^*, where $|u| \leq k$, to the actions shift, accept, error, φ, and reduce i, where i is the number of some production of G;

(2) g maps $N \cup \Sigma$ to $\{error,\varphi\}$ and the names of the tables in the set.

One table is designated to be the initial table.

Note that there is no provision, yet, that the set of tables should form a parser for G. The action of the parser constructed from a set of LR(k) tables $\mathfrak{J}$ for grammar $G = (N,\Sigma,P,S)$ is defined as follows. A configuration of the parser is a pair $(T_0X_1T_1\ldots X_mT_m,w)$, where T_0 is the initial table, $T_0,T_1,\ldots,T_m$ are in $\mathfrak{J}$, $X_1,\ldots,X_m$ are in $N \cup \Sigma$, and w is in Σ^*. Let $T_m = (f,g)$, and let $u = FIRST_k(w)$.

(1) If f(u) = shift, and g(a) = T, where w = aw', then we write $(T_0X_1T_1\ldots X_mT_m,w) \vdash (T_0X_1T_1\ldots X_mT_maT, w')$.

(2) If f(u) = reduce i, production i is $A \rightarrow \alpha$, α is $X_rX_{r+1}\ldots X_m$, and $T_{r-1} = (f',g')$, then $(T_0X_1T_1\ldots X_mT_m, w) \vdash (T_0X_1T_1\ldots X_{r-1}T_{r-1}AT,w)$, where T is g'(A).

(3) If f(u) is error, accept or φ, or if f(x) = reduce i, and $\alpha \neq X_r\ldots X_m$, then there is no configuration C such that $(T_0X_1T_1\ldots X_mT_m,w) \vdash C$. If f(x) = accept, m = 1, $X_1 = S$ and w = e, then the configuration (T_0ST_1, e) is said to be an accepting configuration.

An initial configuration is one of the form (T_0,w). Let $\overset{*}{\vdash}$ be the reflexive and transitive closure of $\vdash$. A configuration C such that $(T_0,w) \overset{*}{\vdash} C$ for some w is said to be accessible. The set $\mathfrak{J}$ is said to be a parser for G if for each w in L(G), $(T_0,w) \overset{*}{\vdash} (T_0ST,e)$ for some T. It is straightforward that the LR(k) tables constructed for an LR(k) grammar as in Section I form a parser for G. However there may be others.

It is possible that given an LR(k) grammar G, we would like to find the "smallest" parser for G. However, there is an important feature of LR(k) parsers which we would like to enforce. As soon as the LR(k) parser of Section I reaches a point where no possible continuation of the input could yield a right sentential form, the parser announces an error. The modifications of [2,3] preserve this property, although each allows the modified parser to perform some reductions when the original parser signals error. The modified parser does not, however, allow a shift after the original has declared an error. Thus, the modifications of [2,3] do not diminish the good error detection and recovery features inherent in Knuth's original parsing algorithm (i.e., the parser of Section I). We will make a definition of equivalent tables which reflects this desire to preserve rapid error detection.

Let $\mathfrak{J}$ and $\mathfrak{R}$ be two sets of LR(k) tables for grammar $G = (N,\Sigma,P,S)$, with initial tables T_0 and R_0, respectively. Let $C_0 \vdash C_1 \vdash \ldots \vdash C_m$ and $D_0 \vdash D_1 \vdash \ldots \vdash D_n$ be sequences of configurations of the parsers constructed from $\mathfrak{J}$ and $\mathfrak{R}$, respectively, such that:

(1) $C_0 = (T_0,w)$ and $D_0 = (R_0,w)$ for some w in Σ^*.

(2) m = n or m < n and there is no configuration C such that $C_m \vdash C$ or n < m and there is no configuration D such that $D_n \vdash D$.

Let $C_m = (T_0X_1T_1\ldots X_rT_r,x)$ and $D_n = (R_0Y_1R_1\ldots Y_sR_s,y)$. We say $\mathfrak{J}$ and $\mathfrak{R}$ are equivalent if for arbitrary sequences as above:

(1) if m = n, then $X_1\ldots X_r = Y_1\ldots Y_s$, x = y, and C_m is an accepting configuration if and only if D_n is an accepting configuration.

(2) if $m \neq n$, then x = y.

Informally, the consequence of the above is that if one parser produces the action error or φ, the other may not shift any more input symbols onto the pushdown list but may reduce before ultimately reporting error.

Our first objective is to determine when the action φ can be truly considered a "don't care," that is, an action which gets exercised. We say a set of tables $\mathcal{J}$ is φ-free if for each accessible configuration $(T_0X_1T_1\ldots X_mT_m, w)$, if $T_m = (f,g)$ and $u = \text{FIRST}(w)$, then:

(1) $f(u) \neq \varphi$, and

(2) if $f(u)$ = shift and the first symbol of w is a, then $g(a) \neq \varphi$, and

(3) if $f(u)$ = reduce i, production i is $A \rightarrow X_r\ldots X_m$ and $T_{r-1} = (f',g')$, then $g'(A) \neq \varphi$.

The following algorithm replaces an error action by φ whenever possible in the tables constructed by Knuth's method.

Algorithm 1: Introduction of φ actions.

Input: The set of Knuth tables for grammar $G = (N,\Sigma,P,S)$.

Output: An equivalent set of tables with error entries replaced by φ, wherever possible.

Method:

(1) Let $T = (f,g)$ be in $\mathcal{J}$. Replace $g(X)$ = error by $g(X) = \varphi$ for each X.

(2) For all u, replace $f(u)$ = error by $f(u) = \varphi$ unless T is the initial table, or there is a table $T' = (f',g')$ in $\mathcal{J}$, a in Σ and v in Σ^* such that

(i) $f'(av)$ = shift,

(ii) $g'(a) = T$, and

(iii) u is v if $|v| < k - 1$, and u is vb for some b in $\Sigma \cup \{e\}$ if $|v| = k - 1$. □

Example 3: Note that if $k = 1$, then $v = e$ in rule (2) of Algorithm 1. Thus, the condition under which error is not replaced by φ reduces to T not being $g'(a)$ for any a and any table (f',g').

Then, the tables of Example 2 become, by Algorithm 1 those shown in Fig. 3.

	a	b	e	A	S	a	b
T_0	S	S	X	T_2	T_1	φ	T_3
T_1	φ	φ	A	φ	φ	φ	φ
T_2	S	S	φ	T_2	T_4	T_5	T_3
T_3	4	4	2	φ	φ	φ	φ
T_4	φ	φ	1	φ	φ	φ	φ
T_5	3	3	X	φ	φ	φ	φ

Fig. 3 LR(1) Tables

The only error entries remaining are in T_0, because it is the initial table, and in T_5, because it appears under a in the goto portion of T_2.

Theorem 1: The set of tables constructed by Algorithm 1 is φ-free and equivalent to the original set.

III. Modification of Table Sets

If a set of tables is φ-free, and two tables have entries which disagree only where one is φ, then the two tables can clearly be identified. A generalization of this idea is the following.

A partition Π on a set of LR(k) tables $\mathcal{J}$ is said to be compatible if whenever $T_1 = (f_1,g_1)$ and $T_2 = (f_2,g_2)$ are in the same block of Π, then

(1) $f_1(u) = f_2(u)$ or $f_1(u) = \varphi$ or $f_2(u) = \varphi$, for each u;

(2) $g_1(X)$ and $g_2(X)$ are in the same block of Π, or $g_1(X) = \varphi$ or $g_2(X) = \varphi$ for each X.

We can merge all tables in a block of a compatible partition by the following algorithm.

Algorithm 2: Merger of compatible tables.

Input: A set $\mathcal{J}$ of φ-free LR(k) tables and a compatible partition Π on $\mathcal{J}$.

Output: A set of LR(k) tables $\mathcal{J}'$ equivalent to $\mathcal{J}$.

Method: $\mathcal{J}'$ consists of one table for each block of Π. The block containing the initial table of $\mathcal{J}$ yields the initial table of $\mathcal{J}'$. Let $\{(f_1,g_1),\ldots,(f_r,g_r)\}$ be a block of Π. Then the table (f,g) constructed from this block has

(i) $f(u) = f_i(u)$ if $f_i(u) \neq \varphi$;

(ii) $g(X)$ is the table constructed from the block of $g_i(X)$ if $g_i(X) \neq \varphi$.

The definition of compatible partition ensures that this construction is consistent. $\square$

Example 4: Let us consider the grammar with productions

(1) $S \to AA$

(2) $A \to a$

(3) $A \to bS$

The φ-free set of tables constructed by Knuth's algorithm and Algorithm 1 is:

	a	b	e	A	S	a	b
T_0	S	S	X	T_2	T_1	T_3	T_4
T_1	φ	φ	A	φ	φ	φ	φ
T_2	S	S	φ	T_5	φ	T_6	T_7
T_3	2	2	X	φ	φ	φ	φ
T_4	S	S	X	T_9	T_8	T_3	T_4
T_5	φ	φ	1	φ	φ	φ	φ
T_6	X	X	2	φ	φ	φ	φ
T_7	S	S	X	T_5	T_{10}	T_3	T_4
T_8	3	3	φ	φ	φ	φ	φ
T_9	S	S	φ	T_{11}	φ	T_3	T_4
T_{10}	φ	φ	3	φ	φ	φ	φ
T_{11}	1	1	φ	φ	φ	φ	φ

Fig. 4 φ-Free Tables

Let Π be the partition with blocks $\{T_0\}$, $\{T_1,T_2\}$, $\{T_3\}$, $\{T_4,T_7\}$, $\{T_5,T_9\}$, $\{T_6\}$, $\{T_8,T_{10}\}$, and $\{T_{11}\}$. If we denote the table for block $\{T_i\}$ by R_i and the table for block $\{T_i,T_j\}$ by R_i, if $i < j$, then the result of Algorithm 2 is shown in Fig. 5. $\square$

	a	b	e	A	S	a	b
R_0	S	S	X	R_1	R_1	R_3	R_4
R_1	S	S	A	R_5	φ	R_6	R_4
R_3	2	2	X	φ	φ	φ	φ
R_4	S	S	X	R_5	R_8	R_3	R_4
R_5	S	S	1	R_{11}	φ	R_3	R_4
R_6	X	X	2	φ	φ	φ	φ
R_8	3	3	3	φ	φ	φ	φ
R_{11}	1	1	φ	φ	φ	φ	φ

Fig. 5 Tables after Compatible Mergers

Theorem 2: The set of table resulting from Algorithm 2 is φ-free and equivalent to the input set of tables.

The second idea for alteration of tables is to postpone certain error checks. If we have table (f,g), and $f(u)$ = error, we could change $f(u)$ to reduce i, if we were sure that:

(1) The right side of production i appears on top of the pushdown list, and

(2) the error would be caught by any table which could appear on top of the pushdown list immediately after the reduction.

In fact (2) is slightly too strong. We would like to know that if (f',g') could next appear on top of the pushdown list, then $f'(u)$ = error. However, we could simultaneously change $f'(u)$ to a reduce action, and equivalence would be preserved.

A few definitions are useful. First we extend the GOTO function to tables and strings of grammar symbols as follows.

If $T = (f,g)$ is an LR(k) table, and $g(X) = T'$, then we say $\underline{GOTO(T,X) = T'}$. We define $\underline{GOTO(T,\alpha)}$, where α is a string recursively as follows.

(1) $GOTO(T,e) = T$

(2) $GOTO(T,\alpha X) = GOTO(GOTO(T,\alpha),X)$.

Let i be production $A \to \alpha$ and let T be a table. We define the function NEXT, by $NEXT(T,i) = \{T' \mid$ there exists table T'' such that $GOTO(T'',\alpha) = T$ and $GOTO(T'',A) = T'\}$. Thus, NEXT(T,i) gives the set of tables that could appear on top of the pushdown list after T calls for a reduce i action.

Let $\mathcal{J}$ be a set of tables. A <u>postponement set</u> for $\mathcal{J}$ is a set $\mathcal{S}$ of <u>triples</u> (T,u,i), where T is in $\mathcal{J}$, u is a terminal string and i is a production number, with the following conditions.

(1) If T = (f,g), then f(u) is error or φ.

(2) If production i is $A \rightarrow \alpha$, and T = GOTO(T′,β), then α is a suffix of β or conversely. If T′ is the initial table, then α must be a suffix of β. (This assures that a reduction of production i will only be called for if α appears on top of the pushdown list.)

(3) If T′ is in NEXT(T,i), and T′ = (f′,g′), then f′(u) is error or φ. (This assures that errors will be caught before a shift, even if (T′,u,j) is also in $\mathcal{S}$ for some j.)

<u>Algorithm 3</u>: Postponement of error checking.

<u>Input</u>: A φ-free set of tables $\mathcal{J}$ and a postponement set $\mathcal{S}$ for $\mathcal{J}$.

<u>Output</u>: A set of tables $\mathcal{J}'$ equivalent to $\mathcal{J}$.

<u>Method</u>:

(1) For each (T,u,i) in $\mathcal{S}$, where T = (f,g), set f(u) to reduce i.

(2) If (T,u,i) is in $\mathcal{S}$, and NEXT(T,i) contains T′ = (f′,g′), set f′(u) = error if it was originally φ and was not changed in step (1).

(3) Let (T,u,i) be in $\mathcal{S}$, and let production i be $A \rightarrow \alpha$. For all T′ = (f′,g′) such that GOTO(T′,α) = T and g′(A) = φ, set g′(A) = error.

Call the resulting set of tables $\mathcal{J}'$. □

<u>Example 5</u>: Let us consider the grammar

(1) $S \rightarrow AS$

(2) $S \rightarrow b$

(3) $A \rightarrow aB$

(4) $B \rightarrow aB$

(5) $B \rightarrow b$

The φ-free set of tables obtained using Algorithm 1 from the Knuth tables for G is shown in Fig. 6.

	a	b	e	S	A	B	a	b
T_0	S	S	X	T_1	T_2	φ	T_4	T_3
T_1	φ	φ	A	φ	φ	φ	φ	φ
T_2	S	S	φ	T_5	T_2	φ	T_4	T_3
T_3	X	X	2	φ	φ	φ	φ	φ
T_4	S	S	X	φ	φ	T_6	T_4	T_8
T_5	φ	φ	1	φ	φ	φ	φ	φ
T_6	3	3	φ	φ	φ	φ	φ	φ
T_7	S	S	X	φ	φ	T_9	T_7	T_8
T_8	5	5	X	φ	φ	φ	φ	φ
T_9	4	4	φ	φ	φ	φ	φ	φ

Fig. 6 φ-Free Tables for G

We can choose to replace the error entries in T_3 with reduce 5, and the error entry of T_8 by reduce 2. That is, we pick a postponement set $\{(T_3,a,5),(T_3,b,5),(T_8,e,2)\}$. Production 5 is $B \rightarrow b$, and GOTO(T_0,b) = GOTO(T_2,b) = T_3. Thus, the entries under B in T_0 and T_2 must be changed from φ to error. Similarly, the entries under S for T_4 and T_7 are changed to error. The resulting set of tables is:

	a	b	e	S	A	B	a	b
T_0	S	S	X	T_1	T_2	X	T_4	T_3
T_1	φ	φ	A	φ	φ	φ	φ	φ
T_2	S	S	φ	T_5	T_2	X	T_4	T_3
T_3	5	5	2	φ	φ	φ	φ	φ
T_4	S	S	X	X	φ	T_6	T_4	T_8
T_5	φ	φ	1	φ	φ	φ	φ	φ
T_6	3	3	φ	φ	φ	φ	φ	φ
T_7	S	S	X	X	φ	T_9	T_7	T_8
T_8	5	5	2	φ	φ	φ	φ	φ
T_9	4	4	φ	φ	φ	φ	φ	φ

Fig. 7 Tables After Postponement of Error Checking.

If we wished, we could now apply Algorithm 2 with a compatible partition grouping T_3 with T_8, T_1 with T_2 and T_5 with T_6. (Other combinations of three pairs are also possible.) The resulting set of tables is given in Fig. 8.

	a	b	e	S	A	B	a	b
T_0	S	S	X	T_1	T_1	X	T_4	T_3
T_1	S	S	A	T_5	T_1	X	T_4	T_3
T_3	5	5	2	φ	φ	φ	φ	φ
T_4	S	S	X	X	φ	T_5	T_4	T_3
T_5	3	3	1	φ	φ	φ	φ	φ
T_7	S	S	X	X	φ	T_9	T_7	T_3
T_9	4	4	φ	φ	φ	φ	φ	φ

Fig 8. Merged Tables

Theorem 3: Algorithm 3 produces a set of tables $\mathcal{J}'$ which is equivalent to $\mathcal{J}$.

IV. DeRemer's Methods

In [3], two subclasses of LR(k) grammars, called SLR(k) and LALR(k), for simple LR(k) and lookahead LR(k), respectively, are defined. In each case, an algorithm which produces sets of LR(k) tables for the grammar of that class are given. The number of tables generated was considerably smaller than the number of Knuth tables. It turns out that the relation of the sets of tables constructed by [1] and [3] can be expressed simply. An application of Algorithm 1 (introduction of φ's), followed by Algorithm 3 (postponement) followed by Algorithm 2 (compatibility) to the Knuth tables yields those generated by DeRemer (with certain error entries made φ).

We will discuss only the SLR(k) method. The LALR(k) method is more general, and can be characterized similarly. For simplicity, from here on, we restrict ourselves to the case $k = 1$. The SLR(1) method can be described as follows.

(1) Construct the set of LR(0) items for the LR(1) grammar $G = (N,\Sigma,P,S)$.

(2) Replace every item of the form $[A \to \beta.,e]$ in set $\mathcal{a}$, by $[A \to \beta.,a]$ for all a in $\mathrm{FOLLOW}^G(a)$, where $\mathrm{FOLLOW}^G(A) = \{a \mid$ there exists a right sentential form αAw such that $a = \mathrm{FIRST}_1^G(w)\}$.

(3) Construct tables from the altered sets of items as in [1]. If the table construction is successful (It is possible that even though G is LR(1), certain action conflicts occur in these sets of items), the set of tables created forms an LR(k) parser for G equivalent to the set generated by [1]. If the method is successful, the grammar G is SLR(1). (Thus, DeRemer's algorithm forms a definition of SLR grammars.)

Theorem 4: The set of tables constructed for SLR(k) grammars by DeRemer's method is equivalent to the Knuth set of tables.

Proof: We will sketch a proof here. First, we observe that the sets of LR(1) items constructed for G by Knuth's algorithm may have two or more sets with common cores. (The core of a set of items is the set of first components, i.e., the core of $[A \to \alpha.\beta,u]$ is $A \to \alpha.\beta$.) Since the second component of an LR(0) item is always e, all distinct sets of LR(0) items have distinct cores. Moreover, the set of cores which appear when the sets of LR(1) items are constructed for G is the same as the set of cores of the sets of LR(0) items for G.

Thus, there is a function f which maps tables constructed by Knuth's algorithm to those constructed by DeRemer's, such that $f(T) = T'$ if and only if T is constructed from a set of LR(1) items having the same core as the set of LR(0) items from which T' was constructed. It is easy to show that f commutes with GOTO. That is, $\mathrm{GOTO}(f(T,X)) = f(\mathrm{GOTO}(T,X))$.

Moreover, if $f(T) = T'$, the only difference in the tables T and T' is that T' may call for reductions when T announces error. This is because T may be constructed from a set of items with $[A \to \alpha.,a]$ but not $[A \to \alpha.,b]$. If b is in FOLLOW(A), then T' will be constructed from a set with both.

However, T may be altered to have the same actions as T' by using Algorithm 3 with the postponement set which consists of all (T,a,i) such that the action of T on a is error, but the action of f(T) on a is reduce i. Then, Algorithm 2 merges all those tables T_1 and T_2 such that $f(T_1) = f(T_2)$. □

Example 6: Let us consider the grammar with productions

(1) S → AA

(2) A → aA

(3) A → b

FOLLOW(S) = {e} and FOLLOW(A) = {a,b}.

We list the LR(1) sets of items (brackets deleted), followed by the set of tables generated therefrom.

α_0	S′ → .S,e
	S → .AA,e
	A → .aA,a/b
	A → .b,a/b
α_1	S′ → S.,e
α_2	S → A.A,e
	A → .aA,e
	A → .b,e
α_3	A → a.A,a/b
	A → .aA,a/b
	A → .b,a/b
α_4	A → b.,a/b
α_5	S → AA.,e
α_6	A → a.A,e
	A → .aA,e
	A → .b,e
α_7	A → b.,e
α_8	A → aA.,a/b
α_9	A → aA.,e

	a	b	e	A	S	a	b
T_0	S	S	X	T_2	T_1	T_3	T_4
T_1	φ	φ	A	φ	φ	φ	φ
T_2	S	S	φ	T_5	φ	T_6	T_7
T_3	S	S	X	T_8	φ	T_3	T_4
T_4	3	3	X	φ	φ	φ	φ
T_5	φ	φ	1	φ	φ	φ	φ
T_6	S	S	X	T_9	φ	T_6	T_7
T_7	X	X	3	φ	φ	φ	φ
T_8	2	2	φ	φ	φ	φ	φ
T_9	φ	φ	2	φ	φ	φ	φ

Fig. 9 LR(1) Tables

The LR(0) items, with ",e" deleted are:

β_0	S′ → .S
	S → .AA
	A → .aA
	A → .b
β_1	S′ → S.
β_2	S → A.A
	A → .aA
	A → .b
β_3	A → a.A
	A → .aA
	A → .b
β_4	A → b.
β_5	S → AA.
β_6	A → aA.

The LR(1) tables constructed from these items by DeRemer's method are shown in Fig. 10.

	a	b	e	S	A	a	b
R_0	S	S	X	R_1	R_2	R_3	R_4
R_1	X	X	A	X	X	X	X
R_2	S	S	X	X	R_5	R_3	R_4
R_3	S	S	X	X	R_6	R_3	R_4
R_4	3	3	3	X	X	X	X
R_5	X	X	1	X	X	X	X
R_6	2	2	2	X	X	X	X

Fig. 10 DeRemer Tables

The function f, which maps $T_0,\ldots,T_9$ to $R_0,\ldots,R_6$ on the basis of the cores of the sets of items from which the tables were generated, is shown in Fig. 11

T	f(T)
T_0	R_0
T_1	R_1
T_2	R_2
T_3	R_3
T_4	R_4
T_5	R_5
T_6	R_3
T_7	R_4
T_8	R_6
T_9	R_6

Fig. 11 Function f

We can apply Algorithm 3 to the T's, with a postponement set consisting of $(T_4,e,3)$, $(T_7,a,3)$, $(T_7,b,3)$, $(T_8,e,2)$, $(T_9,a,2)$ and $(T_9,b,2)$. It is then possible to merge $\{T_3,T_6\}$, $\{T_4,T_7\}$ and $\{T_8,T_9\}$, to obtain tables which are a renaming of the R's.

V. Korenjak's Method

A simple modification of the method of [2] can also be characterized by the operations of merger and postponement. Korenjak's algorithm is, essentially, the following.

(1) Given LR(1) grammar $G = (N,\Sigma,P,S)$, select a splitting set $N' = \{S_1,\ldots,S_m\} \subseteq N$ of nonterminals, including the start symbol, and form grammar $G' = (N,\Sigma \cup \Sigma',P',S_1)$ by replacing symbols in N' on the right side of all productions by corresponding new terminal symbols in $\Sigma' = \{s_1,\ldots,s_m\}$. N', the set so selected may determine whether the algorithm is successful in producing a set of LR(1) tables. Form component grammars G_i, $1 \le i \le m$, from G', by choosing S_i as the start symbol and deleting useless symbols and productions.†

(2) For each component grammar G_i construct the sets of items $\mathcal{S}_i = \{\mathfrak{I}_0^i, \mathfrak{I}_1^i, \mathfrak{I}_2^i, \ldots\}$ such that $\mathfrak{I}_0^i$ is the closure (with respect to G_i) of $\{[S_i \to .\alpha,a] \mid S_i \to \alpha$ is a production in G_i and a is in $\text{FOLLOW}^G(S_i)\}$. However, when taking the closure of an item with respect to G_i we will still use FIRST^G rather than FIRST^{G_i}. For example, if we have added $[A \to \alpha.B\beta,u]$ to $\mathfrak{I}$ and B is not in N', we will also add $[B \to .\gamma,v]$ to $\mathfrak{I}$ where $B \to \gamma$ is in G' and v is in $\text{FIRST}^G(\beta'u)$ where β' is β with all s_h's replaced by S_h's. In this way all lookahead strings will be in Σ^*. Each $\mathfrak{I}_k^i$ in $\mathcal{S}_i$ is the closure of $\text{GOTO}(\mathfrak{I}_k^i,X)$ for some $\mathfrak{I}_j^i$ in $\mathcal{S}_i$ and X in $(N \cup \Sigma' \cup \Sigma)$.

(3) In the first component of all items replace s_i in Σ' on the right side of a production by S_i (the original symbol). Retain the original name for each set of items.

(4) Let $\mathfrak{H}_0 = \{[S_1' \to .S_1,e]\} \cup \mathfrak{I}_0'$. Apply the following augmenting operation to $\mathfrak{H}_0$ and call the resulting set of items $\mathfrak{H}_0$. This $\mathfrak{H}_0$ will be the initial set of items for G.

Augmenting operation: If a set of items $\mathfrak{I}$ contains item $[A \to \alpha.B\beta,a]$ and $B \overset{*}{\underset{G}{\Rightarrow}} S_j\gamma$ for some S_j in N', γ in $(N \cup \Sigma)^*$, then add $\mathfrak{I}_0^j$ to $\mathfrak{I}$. Repeat this process until no new sets of items can be so added to $\mathfrak{I}$.

(5) Now construct $\mathcal{S}$, the set of items for G accessible from $\mathfrak{H}_0$ as follows. Initially, let $\mathcal{S} = \{\mathfrak{H}_0\}$. Then perform step (6) until no new sets of items can be added to $\mathcal{S}$.

(6) Let $\mathfrak{H}$ be in $\mathcal{S}$. $\mathfrak{H}$ can be expressed as $\mathfrak{H} = \mathfrak{I}^0 \cup \mathfrak{I}_{i_1}^{j_1} \cup \mathfrak{I}_{i_2}^{j_2} \cup \ldots \cup \mathfrak{I}_{i_r}^{j_r}$ where $\mathfrak{I}_0$ is either the empty set or $\{[S_1 \to .S_1,e]\}$ or $\{[S_1 \to S_1.,e]\}$.

For each X in $N \cup \Sigma$, let $\mathfrak{I}_0^0 = \text{GOTO}(\mathfrak{I}^0,X)$ and $\mathfrak{I}_{h_n}^{j_n} = \text{GOTO}(\mathfrak{I}_{i_n}^{j_n}, X)$. Let $\mathfrak{H}'$ be the union of $\mathfrak{I}_0^0$ and these $\mathfrak{I}_{h_n}^{j_n}$'s. That is, let $\mathfrak{H}' = \text{GOTO}(\mathfrak{H},X)$. Then apply the augmenting operation to $\mathfrak{H}'$ and call the resulting set of items $\mathfrak{H}'$. Add $\mathfrak{H}'$ to $\mathcal{S}$ if it is not already in $\mathcal{S}$. For the given $\mathfrak{H}$, repeat this process for each X in $N \cup \Sigma$.

(7) When no new set of items can be added to $\mathcal{S}$, construct, if possible, LR(1) tables from $\mathcal{S}$ using Knuth's method. If not possible because of parsing action conflicts, report failure. $\square$

Example 7: Let us consider the grammar G with productions

(1) $S_1 \to S_2S_2$

(2) $S_2 \to aS_2$

(3) $S_2 \to b$

Let $N' = \{S_1,S_2\}$. Then G_1 consists only of the production

(1) $S_1 \to s_2s_2$

and G_2 consists of the two productions

(2) $S_2 \to as_2$

(3) $S_2 \to b$.

The sets of items for G_1 are

† In [2], S_i was not replaced by s_i in the productions of its own grammar. We choose this related approach for its symmetry.

$\mathfrak{I}_0^1$: $S_1 \to .S_2S_2,\ e$

$\mathfrak{I}_1^1$: $S_1 \to S_2.S_2,\ e$

$\mathfrak{I}_2^1$: $S_1 \to S_2S_2.,\ e$

Those for G_2 are

$\mathfrak{I}_0^2$: $S_2 \to .aS_2,\ a/b/e$
$S_2 \to .b,\ a/b/e$

$\mathfrak{I}_1^2$: $S_2 \to a.S_2,\ a/b/e$

$\mathfrak{I}_2^2$: $S_2 \to aS_2.,\ a/b/e$

$\mathfrak{I}_3^2$: $S_2 \to b.,\ a/b/e$

Note that $\mathrm{FOLLOW}^G(S_2) = \{a,b,e\}$.

To compute $\mathcal{H}_0$ we begin with the set of items

$S_1' \to .S_1,\ e$
$S_1 \to .S_2S_2,\ e$

Since $S_2 \overset{*}{\underset{G}{\Rightarrow}} S_2$ and S_2 is in N′ we add to $\mathcal{H}_0$

$S_2 \to .aS_2,\ a/b/e$
$S_2 \to .b,\ a/b/e$

Thus $\mathcal{H}_0 = \{[S_1' \to .S_1,e]\} \cup \mathfrak{I}_0^1 \cup \mathfrak{I}_0^2$. Let us now compute $\mathrm{GOTO}(\mathcal{H}_0,X)$ for X in $\{S_1,S_2,a,1\}$. We obtain

$$\mathcal{H}_1 = \mathrm{GOTO}(\mathcal{H}_0,S_1) = \{[S_1' \to S_1.,e]\}$$

The augmenting operation does not enlarge $\mathcal{H}_1$.

$$\mathcal{H}_2 = \mathrm{GOTO}(\mathcal{H}_0,S_2) = \mathfrak{I}_1^1$$

Since $S_1 \to S_2.S_2$, e is in $\mathfrak{I}_1^1$, the augmenting operation will add $\mathfrak{I}_0^2$ to $\mathcal{H}_2$. Continuing in this fashion, we obtain the following sets of items for $\mathcal{S}$.

$$\mathcal{H}_0 = \{[S_1' \to .S_1,e]\} \cup \mathfrak{I}_0^1 \cup \mathfrak{I}_0^2$$

$$\mathcal{H}_1 = \{[S_1' \to S_1.,e]\}$$

$$\mathcal{H}_2 = \mathfrak{I}_1^1 \cup \mathfrak{I}_0^2$$

$$\mathcal{H}_3 = \mathfrak{I}_0^2 \cup \mathfrak{I}_1^2$$

$$\mathcal{H}_4 = \mathfrak{I}_3^2$$

$$\mathcal{H}_5 = \mathfrak{I}_2^1$$

$$\mathcal{H}_6 = \mathfrak{I}_2^2$$

All these sets of items are consistent (produce no parsing action conflicts), so that we can obtain a set of LR(1) tables for G. These are shown in Fig. 12. Note that Knuth's algorithm would have produced ten tables for this grammar. □

	a	b	e	S_1	S_2	a	b
T_0	S	S	X	T_1	T_2	T_3	T_4
T_1	X	X	A	X	X	X	X
T_2	S	S	X	X	T_5	T_3	T_4
T_3	S	S	X	X	T_6	T_3	T_4
T_4	3	3	3	X	X	X	X
T_5	X	X	1	X	X	X	X
T_6	2	2	2	X	X	X	X

Fig. 11 LR(1) Tables

<u>Theorem 5</u>: The set of tables constructed by Korenjak's algorithm is equivalent to the set of Knuth tables for the same grammar.

<u>Proof</u>: The proof is similar to that for DeRemer's method. One finds a function f which commutes with GOTO, and maps tables from Knuth's set to those from Korenjak's set, preserving the core of the sets of items underlying the two tables. Unlike DeRemer's SLR method, Korenjak's can produce two sets of items with the same cores. □

In a sense, Korenjak's method is a generalization of DeRemer's. The former works whenever the latter does, although DeRemer's algorithm is simpler to implement and works on many naturally occurring grammars. The following relationship is of interest.

Theorem 6: DeRemer's SLR(1) algorithm succeeds in producing LR(1) tables for $G = (N,\Sigma,P,S)$ if and only if Korenjak's algorithm succeeds when the splitting set N' is N (i.e., all nonterminals are made start symbols). The two sets of tables produced in this case are isomorphic under renaming.

We close with a comment that neither Korenjak's nor DeRemer's algorithms represent maximal use of the principles embodied in Algorithms 1 - 3. For example, the grammar

$$\begin{aligned} S_1 &\to S_2AS_2B \\ S_2 &\to aC \\ A &\to b \\ A &\to c \\ B &\to c \\ B &\to d \\ C &\to Cf \\ C &\to f \end{aligned}$$

discussed in [2] takes 18 tables by Knuth's method and 14 by Korenjak's or DeRemer's. It is possible, by Algorithms 1 - 3 to construct an equivalent set of 10 tables.

References

[1] D. E. Knuth, "On the translation of languages from left to right," Inf. Control, Vol. 8, no. 6, 607-639, 1965.

[2] A. J. Korenjak, "A practical method for constructing LR(k) processors," Comm. ACM, vol 12, no. 11, November 1969, 613-623.

[3] F. DeRemer, Practical Translators for LR(k) Languages, Project MAC Report MAC TR-65, October 1969, Cambridge, Mass.

[4] A and U, The Sensuous Compiler, Prentice Hall, Englewood Cliffs, New Jersey, to appear.